Islamic University Of Gaza
Faculty of Engineering
Computer Department

Assembly Language
Discussion
Chapter 3

Created By: Eng. Ahmed M. Ayash
Modified and Presented by: ***Eng. Eihab S. El-Radie***

CHAPTER 3

Assembly Language Fundamentals

- **Basic Elements of Assembly Language**

Q (Yes/No): Is A5h a valid hexadecimal constant?

A No, (a leading zero is required). 0A5h

Q (Yes/No): Does the multiplication operator (*) have a higher precedence than the division operator (/) in integer expressions?

A No, (they have the same precedence)

Q (Yes/No): Must string constants be enclosed in single quotes?

A No, they can also be enclosed in double quotes

Q (True/False): An identifier cannot begin with a numeric digit.

A True

Q (Yes/No): Assembly language identifiers are (by default) case insensitive.

A True

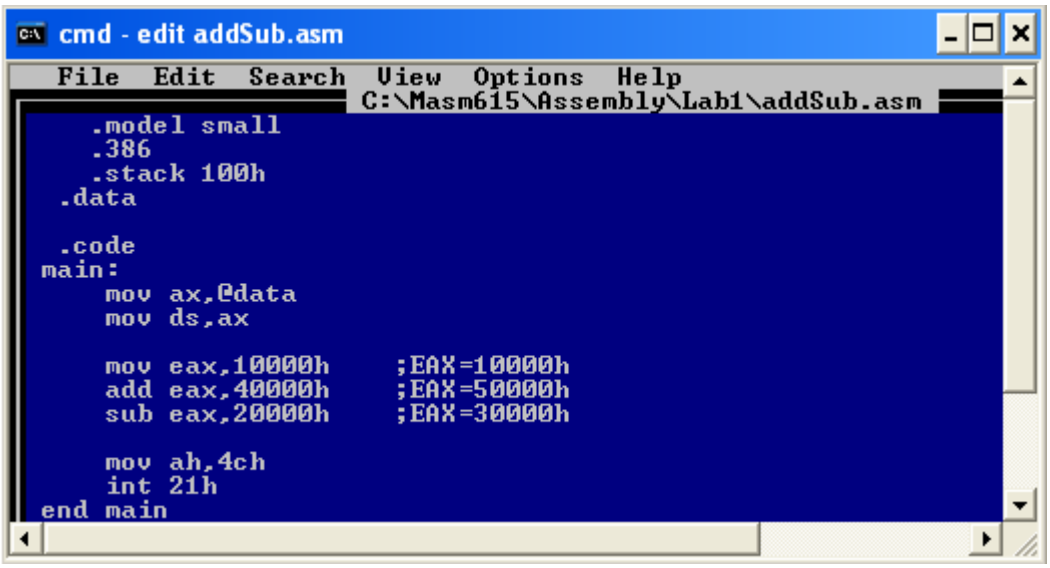
Q (True/False): Assembler directives execute at runtime.

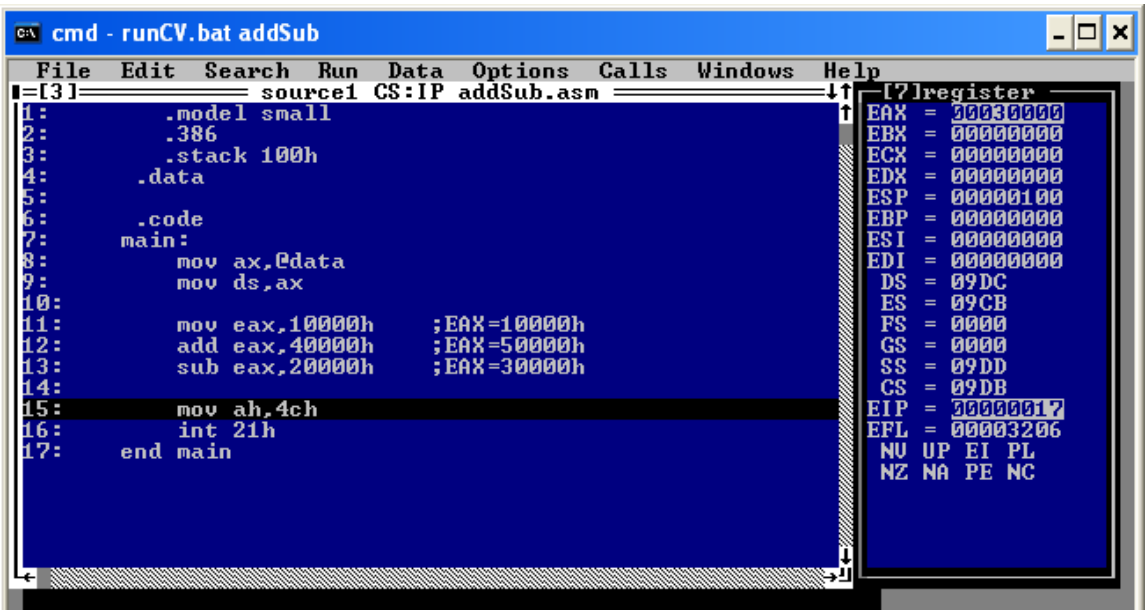
A False.

Q Name the four basic parts of an assembly language instruction.

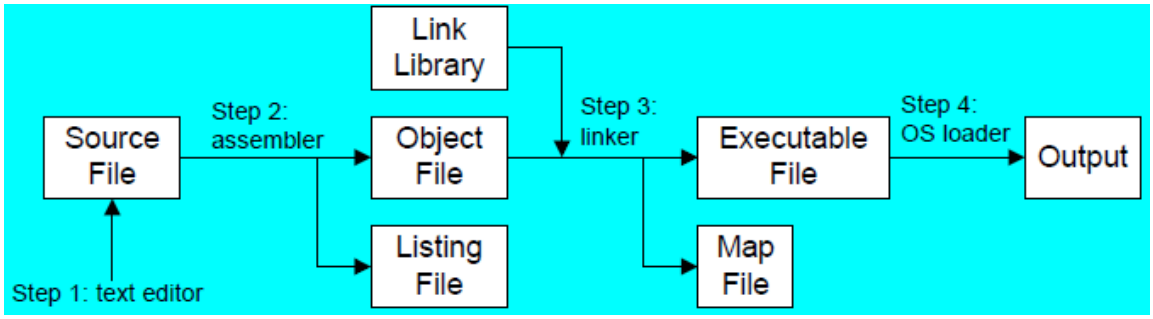
A [label:] mnemonic [operands] [;comment]

➢ Adding and Subtracting Integers Example





➢ Assembling, Linking, and Running Programs



➢ Data Definition Statement

[*name*] *directive initializer* [,*initializer*] . . .

Q Create an uninitialized data declaration for a 16-bit signed integer.

A var1 SWORD ?

Q Create an uninitialized data declaration for an 8-bit unsigned integer.

A var2 BYTE ?

Q Which data type can hold a 32-bit signed integer?

A SDWORD

Q Declare an unsigned 16-bit integer variable named **wArray** that uses three initializers.

A wArray WORD 10,20,30

Q (**3.4.12 p8)** Declare a string variable containing the name of your favorite color. Initialize it as a null terminated string.

A myColor BYTE "blue",0

Q (**3.4.12 p9)** Declare an uninitialized array of 50 unsigned doublewords named **dArray**.

A dArray DWORD 50 DUP(?)

Little Endian Order

x86 processors store and retrieve data from memory using ***little endian*** order (low to high). The least significant byte is stored at the first memory address allocated for the data. The remaining bytes are stored in the next consecutive memory positions. Consider the doubleword **12345678h**. If placed in memory at offset 0000, 78h would be stored in the first byte, 56h would be stored in the second byte, and the remaining bytes would be at offsets 0002 and 0003, as shown in Figure

0000:	78
0001:	56
0002:	34
0003:	12

Some other computer systems use ***big endian*** order (high to low). The following Figure shows an example of **12345678h** stored in big endian order at offset 0:

0000:	12
0001:	34
0002:	56
0003:	78

Symbol Table

Assembler builds a symbol table

- So we can refer to the allocated storage space by name
- Assembler keeps track of each name and its offset
- Offset of a variable is relative to the address of the first variable

Example

```
.DATA
value WORD 0
sum DWORD 0
marks WORD 10 DUP (?)
msg BYTE 'The grade is:',0
char1 BYTE ?
```

Symbol Table

Name	Offset
value	0
sum	2
marks	6
msg	26
char1	40

Symbolic Constants

A symbolic constant (or symbol definition) is created by associating an identifier (a symbol) with an integer expression or some text. Symbols do not reserve storage. They are used only by the assembler when scanning a program, and they cannot change at runtime. The following table summarizes their differences:

	Symbol	Variable
Uses storage?	No	Yes
Value changes at runtime?	No	Yes

Assembler provides three directives:

- = directive
- EQU directive
- TEXTEQU directive

Equal-Sign Directive

The *equal-sign directive* associates a symbol name with an integer expression. The syntax is

name = expression

- Expression is a 32-bit integer (expression or constant)
- may be redefined
- *Name* is called a symbolic constant

Q Declare a symbolic constant using the equal-sign directive that contains the ASCII code (08h) for the Backspace key.

A BACKSPACE = 08h

Q Declare a symbolic constant named **SecondsInDay** using the equal-sign directive and assign it an arithmetic expression that calculates the number of seconds in a 24-hour period.

A SecondsInDay = 24 * 60 * 60

❖ Calculating the Sizes of Arrays and Strings

A better way to declare an array size is to let the assembler calculate its value for you. The **$** operator (*current location counter*) returns the offset associated with the current program statement. In the following example, **ListSize** is calculated by subtracting the offset of list from the current location counter ($):

```
list BYTE 10,20,30,40
ListSize = ($ - list)
```

Rather than calculating the length of a string manually, let the assembler do it:

```
myString BYTE "This is a long string, containing"
         BYTE "any number of characters"
myString_len = ($ - myString)
```

When calculating the number of elements in an array containing values other than bytes, you should always divide the total array size (in bytes) by the size of the individual array elements. The following code, for example, divides the address range by 2 because each word in the array occupies 2 bytes (16 bits):

```
list WORD 1000h,2000h,3000h,4000h
ListSize = ($ - list) / 2
```

Q Write a statement that causes the assembler to calculate the number of bytes in the following array, and assign the value to a symbolic constant named **ArraySize**:

myArray WORD 20 DUP(?)

A ArraySize = ($ – myArray)

Q Show how to calculate the number of elements in the following array, and assign the value to a symbolic constant named **ArraySize**:

myArray DWORD 30 DUP(?)

A ArraySize = ($ – myArray) / 4 OR ArraySize = ($ – myArray) / TYPE DWORD

EQU Directive

- Define a symbol as either an integer or text expression
- Cannot be redefined
- There are three formats:
  - name EQU expression ⟹ SIZE EQU 10*10
  - name EQU <text> ⟹ pressKey EQU <"Press any key to continue...",0>
  - name EQU symbol : *symbol* is an existing symbol name, already defined with = or EQU

TEXTEQU Directive

- Define a symbol as either an integer or text expression.
- Called a text macro
- Can be redefined
- There are three formats:
  - Name TEXTEQU <text> assign any text to name
  - Name TEXTEQU textmacro assign existing text macro
  - Name TEXTEQU %constExpr constant integer expression

For example:

```
continueMsg TEXTEQU <"Do you wish to continue (Y/N)?">
rowSize = 5
count TEXTEQU %(rowSize * 2); constant expression
move TEXTEQU <mov>
setupAL TEXTEQU <move al,count>     ; = setupAL TEXTEQU <mov al, 10>
```

Q Use TEXTEQU to create a symbol named **Sample** for a string constant, and then use the symbol when defining a string variable named **MyString**.

A Sample TEXTEQU <"This is a string">
MyString BYTE Sample

Homework:

1. From Book (7th edition)

Section Review 3.1.11: 4, 6, 7, 8

Section Review 3.3.2: 1, 2, 3, 4, 5

Section Review 3.4.12: 1, 2, 3, 4, 5

2. Others

1. What is the memory byte order, from low to high address, of the following data definition?

   BigVal DWORD 12345678h

2. Write a program that defines symbolic constants for all of the days of the week using = sign. Create an array variable that uses the symbols as initializers.

3. What is the value of the Overflow flag after the execution of code below?

   MOV AL, 88h
   ADD AL, 90h

Quiz Next Week in Chapter3

☺